Abstract:

In this paper we show how the mathematically precise definition of a simple notation for designing and implementing programs involving synchronization of parallel processes introduced by C.A.R. Hoare can be expressed in the COSY formalism. We also indicate how the COSY formalism generalizes the semantic trace model used by Hoare and what might be some advantages of the generalized model. Finally, the COSY notation is used to develop a number of versions of Hoare's examples and the COSY semantic formalism is used to verify that partial or total system deadlock is impossible.

Synchronization of Concurrent Processes without Globality Assumptions

By

P.E. Lauer

TECHNICAL REPORT SERIES

Series Editor: Mr. M.J. Elphick

Number 163
March, 1981

Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

bibliographical details

LAUER, Peter Ernst

Synchronization of concurrent processes without globality assumptions. [By] P.E. Lauer.

Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Laboratory, 1981.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 163).

Added entries UNIVERSITY OF NEWCASTLE UPON TYNE. Computing Laboratory. Technical Report Series, no. 163.

Suggested classmarks (primary classmark underlined)

Library of congress: 001.64404 001.6424
Dewey (17th):
U.D.C. 681.322.06

Suggested keywords

CONCURRENT SYSTEMS
COSY NOTATION
DEADLOCK ANALYSIS
PATH EXPRESSIONS
SYNCHRONIZED PARALLEL PROCESSES
VECTORS OF TRACES
VERIFICATION

Abstract

In this paper we show how the mathematically precise definition of a simple notation for designing and implementing programs involving synchronization of parallel processes introduced by C.A.R. Hoare can be expressed in the COSY formalism. We also indicate how the COSY formalism generalizes the semantic trace model used by Hoare and what might be some advantages of the generalized model. Finally, the COSY notation is used to develop a number of versions of Hoare's examples and the COSY semantic formalism is used to verify that partial or total system deadlock is impossible.

About the author

Dr. P.E. Lauer has been a lecturer in the Computing Laboratory of the University of Newcastle upon Tyne since 1972, before then he was with IBM Laboratory, Vienna.

SYNCHRONIZATION OF CONCURRENT PROCESSES WITHOUT GLOBALITY ASSUMPTIONS

P. E. Lauer

Introduction

The advent of LSI and VLSI technology and microprogramming techniques has greatly increased the number of choices a digital system designer has for decomposing a system into ultimate subsystems which grow considerably more powerful and complex in function as technology progresses. Furthermore, designers have been learning how to combine such subsystems in new ways giving rise to systems which perform the same general functions as earlier systems but with a much greater degree of parallelism and distribution.

We will be concerned with the general problem of the definition and analysis of synchronization in concurrent systems without the assumption of a global nature for such entities as the system clock, the system state, the control, the observer, etc. Systems without such globality assumptions will be called distributed.

If a combination of subsystems is to cooperate coherently to perform a particular system function synchronization is necessary to ensure proper joint behaviours of subsystems with respect to any parts of the system they share. If the resulting system is distributed and capable of parallel behaviours in subsystems, then the required synchronization must be specified without recourse to either a central clock, central control or global state. Many of the conventional formal tools the system designer has at his disposal, such as Automata Theory, are only suitable for adequately expressing sequential systems and they do this under the assumption of a global system state. Hence, designers have had to support such formalisms with additional formal and informal notions when applying them to the specification and analysis of concurrent and distributed systems.

A number of researchers have been seeking for an appropriate extension to the notions of automata theory and other formalisms to permit the inclusion of formal specification and analysis of synchronic properties of systems, such as concurrency, distribution, deadlock and starvation. Among the well developed approaches put forward we mention Petri net Theory (1), the model of synchronized parallel processes of Hoare (2) (3), the calculus of communicating systems of Milner (4), and the concurrent system formalism COSY (7) (9), all of which seem to be converging to an analogous level of abstraction in dealing with the synchronic aspects of systems. Even though there are superficially certain striking similarities in all the approaches mentioned a more careful comparison reveals considerable divergence between some of the approaches with regard to the semantics of concurrency and distribution. Space will not allow a comparison of all these approaches here but we can roughly divide them into two groups depending on the choice made in modelling concurrency.

Petri Nets, COSY } do not reduce concurrency to non-deterministic interleaving

Synchronized Parallel Processes (Hoare), Communicating Systems (Milner) } reduce concurrency to non-deterministic interleaving (at least for the purpose of conceptualisation)

Our research has indicated that the reduction of concurrency to non-deterministic interleaving is adequate for the complete treatment of such problems as proving the absence of partial or total system deadlock, even though the reduction makes the resulting formalism more complex than we feel is required. However, we are convinced that such a reduction makes it even more difficult or even impossible to deal with other system behaviour such as the so called starvation of some subsystem, or to formalize the notion of priority in the presence of concurrency and distribution.

The present paper will introduce the COSY formalism in direct comparison with the model of synchronized parallel processes of Hoare (2) to give these introductory statements more substance.

What is COSY in general?

COSY (the name is derived from Concurrent System) is a formalism intended to simplify the study of synchronic aspects of concurrent and distributed systems where possible by abstracting away from all aspects of systems except those which have to do with synchronization. In other words, if we consider systems as consisting of activities or events we will abstract from any interpretation these events may have when we apply the system and merely consider how occurrences of such uninterpreted atomic events are related to occurrences of other such uninterpreted events in the system. Thus an occurrence of one event may presuppose a corresponding occurrence of another event, or an occurrence of one event may preclude the concurrent occurrence of another event, or several events may occur concurrently (by which we mean "not necessarily in some sequence" rather than "simultaneously", the latter of which appears as an ill-defined concept in our relativistic view of time). Of course it is possible to formally relate interpretations to such uninterpreted systems in order to demonstrate that a system models the behaviour of some actual or possible real world system. We shall take up this point again in a later section.

In the process of developing COSY, we gradually came to formulate what we now consider to be some important requirements that should be possessed by a design methodology for concurrent and distributed systems. Firstly we considered that it was important to encourage the designer to rid himself, as much as possible, of those preconceptions he might have developed as a result of his experience in sequential programming, where it is natural to think of systems as sequential, centralised and synchronous. Indeed, we felt that it would not suit our purpose to base our approach on any kind of modification of a sequential programming language, since in such cases, the designer is obliged to make his understanding of his problem conform to the available programming language constructs. This is undesirable in the sense that the solution of his problem becomes confused with the implementation of the solution, that the structure of the solution may be hidden behind a mass of implementation details which are not in themselves an essential part of it. Another language might serve just as well. This is particularly important if one wishes, as we do, to concentrate on the specification of synchronization properties of systems rather than the problem-oriented functions these systems perform. To base the development (rather than the implementation) of a solution to a systems organisation problem on some specific programming language construct (such as semaphores or monitors) would be rather like basing the business of solving a problem in linear algebra (as opposed to implementing the solution) on FORTRAN as opposed to the appropriate mathematics. In view of this, we chose to specify synchronization properties in terms of the general notion of permitted behaviours in a linguistic or black box manner. A system will be associated with a grammatical type object whose "language" models a set of permitted or required behaviours.

Our system design methodology should therefore be built around a notation which should abstract from non-synchronization semantics of conventional programming constructs and specific synchronization mechanisms.

However, it should be possible to relate the notation both to any other well defined programming notation and to any major theory of concurrent and distributed systems.

The path notation, due to Campbell and Habermann (15), which was designed so that one could state the proper coordination of concurrent processes as the permissible order of execution of operations on shared system objects as part of the object definition, seemed a good starting point for the development of the notation. However, we felt that the original suggestions about how concurrency should be expressed in paths was inappropriate and not a natural or convenient extension of the notion of finite state automaton or regular expression. Furthermore, we felt that the notion of type involving paths was not abstract enough but involved implementation detail which tended to unnecessarily increase the complexity of the problem of system analysis. More discussion of these points can be found in (13) and (8). Hence we developed what we consider a more natural way of expressing concurrency with paths which are called generalized paths (5). Here we have, as required, an abstract notation of a linguistic kind. Indeed a path expression is essentially a collection of regular grammars, each represented as a regular expression. Just a single regular expression determines a set of strings, each of which may be considered as a labelled total order modelling a sequence of executions of the operation which label it, so may a collection of regular expressions determine a set of vectors of strings, where each vector may be considered as a labelled partial order, modelling a non sequential behaviour of operation executions.

The path notation thus satisfies our requirements for a notation for abstractly describing concurrent systems. It also satisfies our requirement that our notation should be easily relatable to other major theories of concurrent systems. The relationship between regular expressions and state machines is well known. There is a relationship between path expressions and Petri-nets which generalises this relationship. In fact every path expression defines a safe net which decomposes into a set of state machines corresponding to the regular expressions from which the path expression is composed. The path expression and corresponding net may each be shown to define exactly the same set of asynchronous behaviours (5).

A number of precise results about this relationship have been obtained and the interested reader is referred to (16) (17) and (6). We only mention that any safe net can be expressed as a behaviourally equivalent basic COSY program and hence we can think of COSY as involving a programming notation alternative to, for example, graphic representations of nets in two dimensional planes. This close correspondence between linguistic features of COSY and those of net theory is deliberate and extends to higher-level and more abstract and general representations of COSY programs and nets. For example, our macro generators, the so called replicators discussed in (9) (7) were developed from some suggestions of H. Genrich, a major contributor to general net theory, and similar generators have been introduced into the net formalism (see (1) p. 523 multiplication of subnets).

An additional requirement is that the methodology should contain facilities for the verification of systems designed in its notation. The semantics in terms of nets, or equivalently, the semantics which regards a path expression as generating a language of vectors, provide a mathematical milieu for the formal definition of systems properties and for the analysis of a path expression to see whether the system it defines possesses such properties. The choice of regular expressions, rather than some more general structure with the power of Turing machines means, in addition, that all such properties are in principle decidable.

So far, then, we have a notation for describing concurrent systems in an abstract manner, clearly related to Net Theory and possessed of a clean behavioural semantics which provides a firm mathematical foundation for verifying behavioural

properties of systems. The further evolution of the COSY methodology was conditioned by other desiderata that we considered essential in a software design environment.

The first is a matter of convenience for the programmer and a facility for generalisation; it is required that regular expressions of arbitrary size and complexity should be capable of being defined in a succinct and illuminating manner, particularly when the regular expression in question has the kind of regularity of structure which might be expressed using iteration. For this, the replicator notation, which contains facilities for the iterative definition of regular expressions involving indexed operation names, was introduced (7) and (9).

It also seemed that the notation should contain facilities for the expression of hierarchy and modularity, in the sense that it contains features which allow the programmer to introduce levels of abstractness into his design and to practice information hiding and which support other techniques of structured programming. Furthermore it should facilitate the design of distributed systems consisting of subsystems which are capable of proper concurrent behaviour without the presupposition of a centralised or global system state or single clock. For these purposes, the notation has been equipped with a class-like construct, for which we use the term system, (7) and (9).

Finally, we need to be able to systematically relate COSY system specifications to implementations in any programming notation. The system construction also contains means for associating specifications with implementations, (7) and (9).

To allow direct comparison with Hoare's treatment (2) of some of the aspects of systems mentioned above, the rest of the present paper will follow the organization of (2) and deal with the same examples used there. Space will only allow us to quote Hoare's main concepts and notation in the appendix of the present paper and we refer the reader to the cited paper for further details.

Basic concepts and notations for sequential systems

Concepts of the COSY model

1. The ultimate components of a system are events each of which is capable of giving rise to a number (possibly zero) of occurrences of that event during any period of (discrete) behaviour of the system.

For example, a vending machine may involve the events:

(E1) insertion of a 5 penny coin
(E2) insertion of a 10 penny coin
(E3) withdrawal of a large packet of biscuits
(E4) withdrawal of a small packet of biscuits
(E5) return of a 5 penny coin

and any number of others.

2. The basis of a system is the set of all events considered to constitute the system.

For example, if we denote the explicitly mentioned events above by their labels, then the basis of the vending machine is

(B1) {E1, E2, E3, E4, E5, ...}

3. A (finite) trace of a sequential system is any (finite) possibly empty sequence of event occurrences which constitutes the history of an actual or potential sequential behaviour of the system from some initial event occurrence up to some final event occurrence.

For example, the following are traces of the vending machine:

(T1) E2.E4.E5 the sequence consisting of an occurrence of E2 followed by an occurrence of E4 followed by an occurrence of E5

(T2) ε the empty sequence of occurrences of events

where the dot "." denotes the juxtaposition of the event occurrences in the order as written left to right.

4. A sequential system is completely determined by the set of all its traces. From the definition of trace it follows that for any system

(i) ε is a trace of the system, and
(ii) If s.t is a trace of the system, then so is s .

Above, (i) indicates that the empty behaviour is a behaviour of any system and it represents the behaviour of the system before any of its events have occurred; and (ii) indicates that any initial segment of a behaviour of a system is itself a behaviour of the system.

In (2) Hoare states that he intends the trace (T1) to correspond to what he calls a successful initial transaction of the vending machine (see appendix). Hence, we might determine the machine by listing the set of all traces corresponding to some successful initial transaction of the machine. Then one could obtain the remaining traces by (ii) above as the set of all prefixes (initial segments) of multiples of successful initial transactions. In fact this is what we have often done in the COSY model where we usually work with systems each of which behaves cyclically when it is operating successfully. In the case of a cyclic sequential system we identify the notion of successful initial history (trace) of the system with the notion of a history (trace) which returns the system to the point which preceded the first occurrence of any of that system's constituent events, or briefly, in the sequential and centralized case, what would be called its initial state.

For example, the vending machine VM could be determined by listing the set of successful initial traces SIT(VM) we intend:

$SIT(VM)=\{E1.E1.E3, E1.E4, E2.E4.E5, E2.E3\}$.

If $\{t_1,\ldots,t_n\}$ is a set of traces, then by $\{t_1,\ldots,t_n\}^*$ we mean the set of all traces of length zero or more obtained by juxtaposing traces t_i and t_j from the set; and if the set of all prefixes of a set of traces T is defined as

$Pref(T) = \{y \mid y.z=x \text{ for some } y, x \in T\}$

then the set of all traces Trace(VM) of the system VM is defined as

$Trace(VM) = Pref(SIT(VM)^*) = SIT(VM)^*.Pref(SIT(VM))$

where "." between two sets of traces denotes the set of all possible juxtapositions of elements from the two sets in the order left to right as written.

In other words

$Trace(VM) = Pref(\{E1.E1.E3, E1.E4, E2.E4.E5, E2.E3\}^*)$
$= \{E1.E1.E3, E1.E4, E2.E4.E5, E2.E3\}^*.\{\varepsilon, E1, E2, E1.E1, E2.E4\}$.

Notation for sequential COSY systems

1. Events belonging to some system are denoted by event symbols in our notation.

For example, to express the vending machine system in COSY we might let the event symbols "5p", "10p", "large", "small" and "5pchange" denote events E1 through E5, respectively.

2. The alphabet of an expression P_S denoting a (sequential) system S, denoted by Alpha (P_S), is the set of event symbols occurring in P corresponding one-to-one to the events in the basis of S. For example,

$Alpha(P_{VM}) = \{5p, 10p, large, small, 5pchange\}$.

3. A sequential system is expressed by writing

path (sequence)*end

where "sequence" is a regular expression formed from the elements of the alphabet of the system by the use of the connectives ";",",", and "*" denoting sequentialization, non-deterministic choice and iteration zero or more times, respectively. More formally a sequential system is expressed by a path expression which is an expression derived from the non-terminal "path" in the little grammar below:

```
path       = path(sequence)*end
sequence   = orelement/orelement;sequence
orelement  = element/element,orelement
element    = event_symbol/element*/(sequence)
```

4. As indicated before, the set of traces of the sequential system expressed by a path expression P is defined in two steps:

4.1 first we define its corresponding set of successful initial traces SIT(P) by:

4.1.1 $SIT(path(sequence)*end) = SIT(sequence)$

The set of successful initial traces of a cyclic path is the set of successful initial traces of its component sequence.

4.1.2 $SIT(orelement_1;\ldots;orelement_n) = SIT(orelement_1).\cdots.SIT(orelement_n)$

The set of successful initial traces of a sequence is the set of all possible juxtapositions of successful initial traces of its constituent orelements taken in the order left to right as written.

4.1.3 $SIT(element_1,\ldots,element_n) = SIT(element_1)\cup\ldots\cup SIT(element_n)$

The set of successful initial traces of an orelement is the set theoretic union of the sets of successful initial traces of its constituent elements.

4.1.4 $SIT(element*) = SIT(element)*$

The set of all successful initial traces of a starred element is the set of traces obtained by juxtaposing members of the set of all successful initial traces of the element any number (possibly zero) of times.

4.1.5 $SIT(event\_symbol) = \{event\_symbol\}$

The set of all successful initial traces of an event symbol is just the singleton set whose only element is the event symbol.

4.1.6 $SIT((sequence)) = SIT(sequence)$

Redundant parentheses do not affect the set of successful initial traces.

4.2 Then we define the set of traces of P by:

$Trace(P) = Pref(SIT(P)*) = SIT(P)*.Pref(SIT(P))$

The set of traces of P is the set of all prefixes of multiples of successful initial traces of P.

Examples

(P1) path (a;b)* end

$Trace(P1) = Pref(SIT(P1)*) = Pref(\{a.b\}*) = \{a.b\}*.\{\varepsilon,a\} = \{\varepsilon,a,a.b,a.b.a,\ldots\}$

(PVM) path((5p;(5p;large),small),(10p;(small;5pchange),large))* end

$SIT(PVM) = \{5p.5p.large,5p.small,10p.small.5pchange,10p.large\}$
and
$Trace(PVM) = SIT(PVM)*.\{\varepsilon,5p,10p,5p.5p,10p.small\}$

A comparison with the notation for synchronized parallel processes introduced by Hoare in (2) (see appendix), shows that at this level of abstraction the only significant divergence is the use of right recursion in the definition of

processes in (2) instead of iteration denoted by the "*" in COSY. When all the right recursions in the definition of a synchronized process P specify the repetition of the same process P then the notation in (2) just reduces to the simple path expressions which form one of the basic components of the COSY formalism (see (5)-(15)). Although, we have not yet developed mechanical rules for this reduction in general our experience with COSY makes us confident that we can also reduce all right recursion to equivalent paths involving only iteration.

Concurrent system model

Concepts of the COSY model

1. The ultimate components of a concurrent system $S = S_1 \ldots S_n$ consisting of sequential subsystems $S_1, \ldots, S_n$, are also events as in the case where S is a single sequential system.

2. The basis of a concurrent system $S = S_1 \ldots S_n$ is the set theoretic union of the bases of its constituent subsystems. More formally

$Bas(S) = Bas(S_1) \cup \ldots \cup Bas(S_n)$

3. A vector of (finite) congreable traces of a concurrent system $S = S_1 \ldots S_n$ is a vector whose i-th component is a (finite) trace of the i-th subsystem S_i for all $1 \leq i \leq n$, and in which the traces of all subsystems agree about the number and order of occurrences of events they share. In other words a vector of (finite) traces of S is any vector of (finite, possibly empty) sequences of event occurrences which constitutes the history of an actual or potential concurrent behaviour of the system from some (possibly concurrent) event occurrences up to some (possibly concurrent) event occurrences. The i-th subhistory of the vector of histories is a history of an actual or potential sequential behaviour of the i-th sequential subsystem from some event occurrence to some event occurrence. But in addition, if several subsystems involve the same event then their respective traces have to agree about the number and order of occurrences of the event in which they coincide. Hence we talk about a vector of congreable traces.

We formally characterize vectors of congreable traces by introducing the notions of projection, event vector, and juxtaposition of event vectors. Next we associate an event vector with each event in the basis Bas(S) of $S = S_1 \ldots S_n$, and call the set of event vectors corresponding to S the vector basis of S, VBas(S) for short. As in the case of "Bas(S)*" we mean by "VBas(S)*" the set of all vectors of traces of length zero or more obtained by juxtaposing event vectors from VBas(S). Finally, the set of vectors of congreable traces corresponding to S, denoted VTrace(S), is that part of the cartesian product of the sets of traces of the subsystems for which we can also find a corresponding element of the set of all vectors of traces.

4. A concurrent system is completely determined by the set of all its vectors of congreable traces. From the definition of vector of traces it follows again that

(i) $\underline{\varepsilon} = (\varepsilon, \ldots, \varepsilon)$ is a trace of the system, and

(ii) If $\underline{s}.\underline{t}$ is a trace of the system, then so is $\underline{s}$;

where: if $\underline{s} = (s_1, \ldots, s_n)$ and $\underline{t} = (t_1, \ldots, t_n)$ are traces

then $\underline{s}.\underline{t} = (s_1, \ldots, s_n).(t_1, \ldots, t_n) = (s_1.t_1, \ldots, s_n.t_n)$.

is a vector of traces, then $\underline{s}$ is a vector of traces of the system. Note that juxtaposition of vectors of traces is defined as component wise juxtaposition of corresponding traces. $\underline{\varepsilon}$ stands for the vector of empty traces.

We will now be slightly more formal but we will use examples to illustrate the formalism. The vector basis of a system $S = S_1 \ldots S_n$ is defined to be

$VBas(S) = \{\underline{a} \mid \exists a \in Bas(S) \text{ and } \underline{a} = (proj_1(a),\ldots,proj_n(a))\}$

where for $1 \leq i \leq n$:

$$proj_i(a) = \begin{cases} a \text{ if } a \in Bas(S_i) \\ \epsilon \text{ otherwise} \end{cases}$$

For example, assume we have a concurrent system VMC consisting of the vending machine VM and a customer CUST, where the basis and the traces of CUST are:

$Bas(CUST) = \{E1,E2,E3,E4\} = Bas(VM) - \{E5\}$

where '-' indicates set theoretic subtraction, and

$Trace(CUST) = \{E1.E3,E2.E3,E4,E3\}^*.\{\epsilon,E1,E2\}.$

Then the basis of the combined system VMC is

$Bas(VMC) = Bas(VM) \cup Bas(CUST) = Bas(VM)$

and the vector basis of VMC is

$VBas(VMC) = \{(E1,E1),(E2,E2),(E3,E3),(E4,E4),(E5,\epsilon)\}.$

Finally, implicitly characterized, the set of vectors of traces of VMC would be

$VTrace(VMC) = (Trace(VM) \times Trace(CUST)) \cap VBas(VMC)^*$

and explicitly it would yield

$VTrace(VMC) = \{(E2.E3,E2.E3)\}^*.\{\underline{\epsilon},(E1,E1),(E2,E2)\}$

which in this case happens to be equivalent to

$VTrace(VMC) = \{\underline{a} \mid \exists a \in Trace(VM) \cap Trace(CUST) \text{ and } \underline{a} = (a,a)\}.$

Note that our vector traces tell us that the only successful initial (cyclic) congreable behaviour in the system is (E2.E3,E2.E3) which corresponds to an occurrence of the event "insertion of a 10 penny coin" followed by an occurrence of the event "withdrawal of a large packet of biscuits". Furthermore, the fact that (E1,E1) is not a prefix of any successful initial (cyclic) congreable behaviour of the system indicates that the system will deadlock after the occurrence of the event "insertion of a 5 penny coin", as was Hoare's intent in constructing this example in (2). The deadlock occurs because after the insertion of a 5 penny coin the customer can only ask for a large packet of biscuits but the vending machine will only allow this after a previous insertion of a 10 penny coin or the successive insertion of two 5 penny coins.

Before we consider more examples and discuss our modelling of concurrent events we introduce the notation for expressing concurrent systems.

Notation for concurrent COSY systems

1. Given a set of sequential paths $P_1,\ldots,P_n$ where for $1 \leq i \leq n$, P_i denotes some sequential system S_i then a concurrent system $S = S_1 \ldots S_n$ is expressed in COSY by writing a concurrent (or generalised) path expression

system $P_1 \ldots P_n$ endsystem

2. The alphabet Alpha (P_S) of a concurrent path expression P_S corresponding to system S is the set theoretic union of the alphabets of its constituent P_i, $1 \leq i \leq n$:

$Alpha(P_S) = Alpha(P_1) \cup \ldots \cup Alpha(P_n).$

For example, we can express the system VMC by the concurrent path expression (PVMC)

```
system
(PVM) path (5p;(5p;large),small),(10p;(small;5pchange),large) end
(PCUST) path large,small,(10p;large),(5p;large) end
endsystem
```

Note that we have omitted the outermost parentheses and the "*" in writing paths but they are still assumed to be cyclic as before. SIT(PVM) and Trace (PVM) have been given earlier.

Trace(CUST) = {large,small,10p.large,5p.large}*.{ε,5p,10p}

Analogously to the construction for concurrent systems in the previous section we obtain the vectors of traces of P_S from vectors of event symbols. Hence,

VTrace(P_S) = (VTrace(P_1)x...xVTrace(P_n)) ∩VAlpha(P_S)*

and

VTrace(PVMC) = {(10p.large,10p.large)}*.{ε,(5p,5p),(10p,10p)}

We will from now on ignore the difference between events of the system being expressed and event symbols denoting them, since we will be concerned with properties of systems, such as absence of (total) deadlock and absence of partial deadlock, which can be studied solely in terms of the uninterpreted event symbols and their interconnection by the connectives ";",","," and "*" in sequential paths and by coincidence of event symbols in concurrent paths. The difference between events and event symbols must be made when one studies whether a system specification in COSY accurately models some real system. We have dealt with the latter topic in references (9)(12) and (13).

In our researches we have come to call a system which is incapable of total system deadlock a deadlock-free system and one which is incapable even of partial system deadlock an adequate system. We now define these notions in our formal model.

(DF) P is deadlock-free if and only if ∀x∈VTrace(P)∃a∈Alpha(P):x.a∈VTrace(P)

(A) P is adequate if and only if

∀x∈VTrace(P)∀a∈Alpha(P)∃y∈VAlpha(P)*:x.y.a∈VTrace(P)

where a is the vector of events corresponding to a.

(DF) says that P is deadlock-free exactly when for every vector of (congreable) histories of P there always exists at least one event of P such that its occurrence could coincidentally extend that vector of histories in all components corresponding to subsystems involving that event.

(A) says that P is adequate exactly when for every vector of (congreable) histories of P and any event of P, there exists an extension of the vector of histories after which the occurrence of the event could coincidentally extend all components corresponding to subsystems involving that event.

According to these definitions PVMC is neither deadlock-free nor adequate since the vector of traces "(5p,5p)" cannot be coincidentally extended by the occurrence of any event of PVMC according to its structure.

The following two modifications of Hoare's vending machine example will provide us with an opportunity to discuss how one might use our model to demonstrate that a system is deadlock-free or adequate.

(PVMC1)
System
(PVM)path(5p;(5p;large),small),(10p;(small;5pchange),large)end
(C1)path 5p;small end
(C2)path 10p;large end
endsystem

VTrace(PVMC1) = {(5p.small,5p.small,ε),(10p.large,ε,10.large)}*.
{ε,(5p,5p,ε),(10p,ε,10p)}

Examination of the set of prefixes Pref(VSIT(PVMC1)), that is the second set above, shows that every prefix can be extended by an occurrence of at least one

event. Hence, PVMC1 is deadlock-free. However, it is not adequate since examination of VTrace(PVMC1) indicates that certain possible behaviours of the vending machine on its own, such as "5p.5p.large" or "10p.small.5pchange" can never occur in the system involving the two customers with their restricted behaviour. Hence, for no $x \in$ VTrace(PVMC1) does there exist a continuation $y \in$ VAlpha(PVMC1)* such that, for example $x.y.$ 5pchange $\in$ VTrace(PVMC1). Hence PVMC1 is not adequate.

(PVMC2)
system
(PVM1) path (5p;small),(10p;large) end
(C1) path 5p;small end
(C2) path 10p;large end
endsystem

VTrace(PVMC2) = VTrace(PVMC1)

but PVMC2 is not only deadlock-free but adequate since the definition of adequacy is satisfied as the reader can verify by an exhaustive consideration of the set of vectors of traces.

But so far our combined system consisting of vending machine and customers is totally sequential, though the order in which customers will succeed with successful initial transactions is by non-deterministic choice. To obtain an example which would allow us to discuss concurrency modelling and to explain one of our theorems which simplifies the proof of adequacy, we specify a vending machine which may be used by two customers concurrently, that is, a machine that has distinct slots for 5p and 10p coins and two distinct points for extraction of small and large packets of biscuits.

(PVMC3)
system
(P1) path 5p;small;plunk end
(P2) path 10p;large;plonk end
(P3) path 5p;small end
(P4) path 10p;large end
endsystem

} vending machine
} customers

In PVMC3 "plunk" and "plonk" denote the sounds made by a small and large packet of biscuits dropping out of the machine, respectively. This kind of machine is called the noisy machine by Hoare in (2), and involves events of the machine which do not need cooperation of customers to occur.

The event vectors of VAlpha(PVMC3) are:

5p = (5p, ε, 5p, ε) 10p = (ε, 10p, ε, 10p)
small = (small, ε, small, ε) large = (ε, large, ε, large)
plunk = (plunk, ε, ε, ε) plonk = (ε, plonk, ε, ε).

Considering any two event vectors on a single line above we see that each such pair has at most one non-empty event as corresponding components. This means that the events corresponding to any such pair of event vectors may be executed concurrently relative to each other and formally this is modelled by the fact that the juxtaposition of the event vectors commutes. In general, we define for distinct events a and b and any vector of traces x of P: a and b are concurrent relative to each other after history x if $x.a \in$ VTrace(P) and $x.b \in$ VTrace(P) and $a.b = b.a$.

For example, in PVMC3, 5p and 10p are concurrent relative to each other initially, Since $\varepsilon.5p \in$ VTrace(PVMC3) and $\varepsilon.10p \in$ VTrace(PVMC3) and
$5p.10p = (5p, \varepsilon, 5p, \varepsilon).(\varepsilon, 10p, \varepsilon, 10p) = (5p, 10p, 5p, 10p) = (\varepsilon, 10p, \varepsilon, 10p).(5p, \varepsilon.5p, \varepsilon) = 10p.5p.$
That is, an occurrence of 5p and a concurrent occurrence of 10p can begin a concurrent behaviour of PVMC3 . These concurrent occurrences need not be interleaved, for example, as after arbitration relative to a single clock.

The system must be capable of functioning correctly even without the assumption that concurrent occurrences of 5p and 10p must be "observable". Consideration of PVMC3 indicates that the events 5p and 10p occur in no single path and hence there is no sequential subsystem to which both belong. Their occurrences are to be thought of as independent and should not in general be reduced to arbitrary interleaving. We will return to this point later.

If we reconsider PVMC3 we see that none of the event symbols occur more than once in a single sequential path and that the comma "," is nowhere used. Concurrent paths of this form are called GE_0-paths (see(5) and (9)), and in our formal theory of COSY systems there exists a theorem which yields a more effective way of deciding whether a GE_0-path is adequate or not, than by considering all histories of possible behaviours of the path, as we have done in our analysis of the examples so far. In fact, the theorem says that it is sufficient and necessary to find a single history of the whole system P such that each of its components is an element of the set of successful initial traces of the corresponding subsystem P_i. Formally, we say P is adequate if and only if

$$\exists \underline{x} \in VTrace(P)\ \forall i \in \{1,\ldots,n\} : [\underline{x}]_i \in SIT(P_i),$$

where $[\underline{x}]_i$ denotes the i-th component of the vector $\underline{x}$.

Reconsidering PVMC3 it is easy to verify that

$\underline{vmc}$ = (5p.small.plunk,10p.large.plonk,5p.small,10p.large)

is a vector of traces of PVMC3 by noting that each component is a trace of the corresponding path P_i, $1 \leq i \leq 4$, and by exhibiting $\underline{vmc}$ as a juxtaposition of vectors of events of PVMC3; for example

$\underline{vmc}$ = $\underline{5p}.\underline{small}.\underline{plunk}.\underline{10p}.\underline{large}.\underline{plonk}$.

The fact that $\underline{vmc}$ satisfies the theorem is established by showing that

$[\underline{vmc}]_1$ = 5p.small.plunk $\in$ SIT(P1)

$[\underline{vmc}]_2$ = 10p.large.plonk $\in$ SIT(P2)

$[\underline{vmc}]_3$ = 5p.small $\in$ SIT(P3)

$[\underline{vmc}]_4$ = 10p.large $\in$ SIT(P4)

We have obtained a number of theorems of this nature for considerably larger classes of programs, and we have applied these results to the verification of numerous operating system strategies for sharing resources in concurrent and distributed computer systems (see references (5)-(14), and (16),(17)).

Comparisons

We have now introduced the reader to the COSY formalism and indicated how one can specify concurrent systems which can be verified with regard to such global behavioural properties as absence of deadlock and adequacy. Along the way we have also briefly pointed out similarities and differences to Hoare's notation and model for synchronized parallel processes (2)(3). This final section of the paper will add a few more remarks of a comparative nature but a thorough and precise comparison of COSY with Hoare's work in (2) and (3) is beyond the scope of the present paper.

Briefly recall what we have already noted about the two approaches. Hoare argues for an interleaving model of concurrency and we argue for a non-interleaving model of concurrency. As far as the notations are concerned, Hoare's notation in (2) can be considered to be equivalent in expressive power to concurrent path expressions as introduced in (5) if we ignore the abort process. Finally, we make a stronger distinction between the system being described or specified and the notation used to specify the system. The first is an actually or potentially existing and dynamic system, not necessarily having a

linguistic nature, and the latter always being a linguistic entity. We feel that there is need to maintain a distinction even at the level of informality of the papers under discussion if the possibility of misinterpretation of statements concerning systems involving concurrency and distribution is to be avoided.

We will take up the above points in some more detail now.

Interleaving or non-interleaving

In general, we can give several reasons for having a formal non-interleaving model of concurrency in addition to an interleaving model, even though most workers in this problem area still feel that the latter is sufficient and convenient.

1. In order to demonstrate that an interleaving model of concurrency can fully model all concurrent systems that can be modelled by a non-interleaving model of concurrency, it is necessary to give a formal notion of a non-interleaving model and to prove the two models equivalent in some sense.

2. There is strong evidence that the two approaches are not equivalent when one has to express behaviours of concurrent systems which are maximal in the sense, for example, that everything that could have happened has happened. Such maximal behaviours are used to prove properties of a compound system such as absence of starvation of some subsystem due to, for example, unfair scheduling or malicious cooperation on the part of other subsystems. The latter kind of starvation is possible for the philosophers in Hoare's paper (2) though no deadlock can occur.

3. Even for those problems where such an equivalence can be demonstrated it is still arguable that the non-interleaving model yields a simpler mathematics than the interleaving model.

To make these points clearer consider the very simple system of concurrent paths below:

(PS1) system path a end path b end endsystem

(PS2) system path a end path b end path a,b end endsystem

VAlpha(PS1) = $\{(a,\varepsilon),(\varepsilon,b)\}$

and $\underline{a}.\underline{b} = \underline{b}.\underline{a}$ means that a is concurrent to b,

but

VAlpha(PS2) = $\{(a,\varepsilon,a),(\varepsilon,b,b)\}$ and

$\underline{a}.\underline{b} = (a,\varepsilon,a).(\varepsilon,b,b) = (a,b,a.b) \neq (a,b,b.a) = (\varepsilon,b,b).(a,\varepsilon,a) = \underline{b}.\underline{a}$

which means that a is not concurrent with b.

If one uses an interleaving model of concurrency the two systems are the same unless one introduces further complexity into the system by dividing concurrent events into two events each, one for the beginning of an occurrence of the event and one for the ending of an occurrence of the event. We shall explain further. For certain purposes we have used an interleaving definition of concurrent composition of paths in the period 1975-1977 (see (5),(6),(8),(11),(17)) which is identical with Hoare's definition of parallel composition of processes P and Q, denoted by "(P||Q)", in (2) p. 110. We give our form of the definition for direct comparison with the vector model of the present paper. The interleaving semantics of the concurrent path $P = P_1 \ldots P_n$ is:

(IS) $\text{Trace}(P) = \{x \in \text{Alpha}(P)^* \mid \forall i \in \{1,\ldots,n\}\ \text{proj}_i(x) \in \text{Trace}(P_i)\}$

where proj_i is defined for single events as before and furthermore for

$a_1.\ldots.a_n \in \text{Alpha}(P)^*$ we define

$\text{proj}_i(a_1.\ldots.a_n) = \text{proj}_i(a_1).\ldots.\text{proj}_i(a_n)$ for all $1 \leq i \leq n$.

If one divided every operation of PS1 and PS2 into begin and end events we would obtain:

(PS1') system path a_begin;a_end end path b_begin;b_end end endsystem

and

(PS2') system path a_begin;a_end end
path b_begin;b_end end
path (a_begin;a_end),(b_begin;b_end) end
endsystem

then one can see that according to PS1'

a_begin.b_begin.a_end.b_end

is a possible (concurrent) behaviour of PS1', whereas according to PS2' it is not since its third path forbids any overlapping of the events a and b. However, we feel that this splitting of events enforced by the interleaving semantics is an unnecessary over complication.

Another overcomplication of the semantical tools enforced by the interleaving model is the fact that one has to work with equivalence classes of traces where one would be working with a single vector in the vector of traces approach we are advocating. Hence, the vector vmc of the previous section represents the entire equivalence class of traces which differ only with respect to permutation of the interleaving of concurrent events.

Finally, the reader will have noticed that we have enclosed the word "finite" in parentheses in many of our definitions because Hoare did so in the corresponding definitions in (2). We can drop the word finite from all our definitions and all true statements made about the model will be true as before. However, the interleaving model does not generalize as easily if at all. Suppose we wanted to say formally what constitutes a maximal behaviour of PS1, that is, a behaviour in which everything that could have happened has happened. In the interleaving model a trace is represented by a sequence of event occurrences (or symbol occurrences) and a trace is maximal if it is not the prefix of any trace longer than it. Hence, the following traces of PS1 are maximal traces:

aaa... infinite number of occurrences of a

baaa... occurrence of b followed by an infinite number of occurrences of a

bbb... infinite number of occurrences of b

but none of these corresponds to a maximal behaviour of PS1 which consists for example of an infinite number of a's followed by an infinite number of b's which cannot be represented as a trace.

If however we go to the non-interleaving model of vectors of traces then the following vectors of traces corresponding to the above examples of maximal traces are not vectors of maximal traces

(aaa..., ε)

(aaa...,b)

(ε,bbb...) ...

For example the first is a vector of prefixes of the second and hence is not maximal. But (aaa...,bbb...), which cannot be represented as an infinite sequence of a's followed by an infinite sequence of b's in the interleaving model, is a vector of maximal traces and is precisely the unique formal expression denoting the maximal behaviour of PS1.

Conclusion

In this short paper we have tried to show how the mathematically precise definition of a simple notation for designing and implementing programs involving synchronization of parallel processes introduced by C.A.R. Hoare in (2) can be expressed in the COSY formalism. We have also indicated how the COSY formalism generalizes the semantic trace model used by Hoare and what might be some advantages of the generalized model. Finally, we used the COSY notation to develop a number of versions of Hoare's examples and used the COSY semantic formalism to verify that partial or total system deadlock is impossible.

Acknowledgements

I would like to acknowledge my indebtedness to Tony Hoare with regard to my understanding of concurrency, in particular for his repeated creation of definitive concurrent programming language concepts. His ideas in (2) and (3) were, as always, highly stimulating and challenging to me and the present paper represents an initial attempt to formulate the similarities and differences of the two approaches. Acknowledgement is due to John Cotronis who has given much of his time to discuss these topics with me and with whom I am producing a detailed comparison of the work going on at Oxford and Newcastle upon Tyne. The work reported in this paper was supported by a grant from the Science Research Council of Great Britain. My indebtedness to the thorough and fruitful researches of Eike Best and Mike Shields should be evident from the appearence of their names in the Bibliography. The author would like to thank Mrs. Joan Armstrong for her patience and efficiency in preparing this typescript.

References

1. Net Theory and Applications: Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg 1979 (Ed. W. Brauer) Lecture Notes in Computer Science 84, Springer Verlag 1980.

2. Hoare, C.A.R.: Synchronisation of Parallel Processes. In: Advanced Techniques for Microprocessor Systems. (Ed. F.K. Hanna), Peter Peregrinus Ltd. 1980.

3. Hoare, C.A.R.: Communicating Sequential Processes. In: On the construction of programs. (Eds. McKeag and MacNaghten) Cambridge University Press 1980.

4. Milner, R.: A Calculus of Communicating Systems. Lecture Notes in Computer Science 92, Springer Verlag 1980.

5. Lauer, P.E., Campbell, R.H.: Formal semantics for a class of high level primitives for coordinating concurrent processes. Acta Informatica 5, 247-332 (1975).

6. Lauer, P.E., Shields, M.W., Best, E.: The design and certification of asynchronous systems of processes. Proc. of EEC Advanced Course on Abstract Software Specification, Lyngby, Jan. 22 - Feb. 2, 1979. Lecture Notes in Computer Science, No. 86, Springer Verlag, 1979, pp. 451-503.

7. Lauer, P.E., Torrigiani, P.R., Shields, M.W.: COSY: a system specification language based on paths and processes. Acta Informatica, Vol. 12, pp. 109-158, 1979.

8. Lauer, P.E., Shields, M.W.: Abstract specification of resource accessing disciplines: adequacy, starvation, priority and interrupts. SIGPLAN Notices, Vol. 13, No. 12, Dec. 1978.

9. Lauer, P.E., Shields, M.W.: COSY An environment for development and analysis of concurrent and distributed systems. In: Software Engineering Environments (Ed. H. Hünke) North Holland 1981.

10. Shields, M.W.: Adequate Path Expressions. Proc. Symp. on the Semantics of Concurrent Computation, Evian-les-Bains, July 2-4, 1979. Lecture Notes in Computer Science Vol. 70, Springer Verlag 1979.

11. Shields, M.W., Lauer, P.E.: On the abstract specification and formal analysis of asynchronization properties of concurrent systems. Proc. of Int. Conf. on Mathematical Studies of Information Processing. Aug. 23-26, Kyoto, 1978. Lecture Notes in Computer Science 75, Springer Verlag 1979, pp. 1-32.

12. Shields, M.W., Lauer, P.E.: A formal semantics for concurrent systems. Proc. 6th Int. Colloqu. for Automata, Languages and Programming, July 16-21, 1979 Graz, Lecture Notes in Computer Science, 71, Springer Verlag, 1979, pp. 569-584.

13. Shields, M.W., Lauer, P.E.: Verifying concurrent system specification in COSY. Proc. 8th Symposium on Mathematical Foundations of Computer Science, Aug. 31 - Sept. 6, 1980, Poland. Lecture Notes in Computer Science, No. 88, Springer Verlag 1980, pp. 576-586.

14. Lauer, P.E., Torrigiani, P.R. Devillers, R.: A COSY Banker: Specification of highly parallel and distributed resource management. Proc. 4th International Symposium on Programming, Paris, April 22-24, 1980. Lecture Notes in Computer Science 83, Springer Verlag 1980, pp. 223-239.

15. Campbell, R.H., Habermann, A.N.: The specification of process synchronization by path expressions. Lecture Notes in Computer Science V. 16, Springer Verlag, pp. 89-102.

16. Best, E.: Adequacy of Path Programs In: Net Theory and Applications: Proceedings of the Advanced Course on General Net Theory of Processes and Systems. Hamburg, 1979. (Ed. Prof. Wilfried Brauer). Lecture Notes in Computer Science 84, Springer Verlag 1980.

17. Lauer, P.E., Shields, M.W., Best, E.: Formal Theory of the Basic COSY Notation. The Computing Laboratory, University of Newcastle upon Tyne, Tech. Rep. Series No. 143, November 1979.

Appendix: Verbatim Quotes of Definitions from Reference (2)

Concepts

1. The ultimate constituent of our model is a symbol, which may be intuitively understood as denoting a class of event in which a process can participate.

a) "5p" denotes insertion of a coin into the slot of a vending machine VM.

b) "large" denotes withdrawal from VM of a large packet of biscuits.

2. The alphabet of a process is the set of all symbols denoting events in which that process can participate.

c) {5p, 10p, large, small, 5pchange} is the alphabet of the vending machine VM.

3. A trace is a finite sequence of symbols recording the actual or potential behaviour of a process from its beginning up to some moment in time.

d) <10p, small, 5pchange> is a trace of a successful initial transaction of VM.

e) <> (the empty sequence) is a trace of its behaviour before its first use.

4. A process P is defined by the set of all traces of its possible behaviour. From the definition of a trace, it follows that for any process P,

   <> is in P (i.e. P is non-empty)

   If st (the concatenation of s with t) is in P then so is s by itself. (i.e. P is prefix-closed).

Notations

1. The process ABORT is one that does nothing ABORT = {<>}

2. If c is a symbol and P is a process, the process $(c \rightarrow P)$ first does "c" and then behaves like the process P.

   $$(c \rightarrow P) = \{<>\} \cup \{<c>s \mid s \text{ is in } P\}$$

   where <c> is the sequence consisting solely of the symbol c. By convention the arrow associates on the right, so that

   $$c \rightarrow d \rightarrow P = c \rightarrow (d \rightarrow P).$$

3. The process $P \,[]\, Q$ behaves either like the process P or like the process Q; the choice will be determined by the environment in which it is placed.

   $P \,[]\, Q = P \cup Q$ (normal set union)

   By convention $\rightarrow$ binds more tightly than $[]$, so that

   $$c \rightarrow P \,[]\, d \rightarrow Q = (c \rightarrow P) \,[]\, (d \rightarrow Q).$$

4. The alphabet of a process P will be denoted by $\alpha(P)$. Usually we will assume that the alphabet of a process is given by the set of all symbols occurring in its traces.

5. We shall use recursive definitions to specify the behaviour of long-lasting processes. These recursions are to be understood in the same sense as the recursive equations of (say) a context-free grammar expressed in BNF.

Examples

f) $P = (a \rightarrow b \rightarrow P)$

$= \{\langle\rangle, \langle a\rangle, \langle a,b\rangle, \langle a,b,a\rangle, \ldots\}$

g) $VM = (5p \rightarrow (5p \rightarrow (large \rightarrow VM \;\square\; 5p \rightarrow ABORT)$

$\square\; small \rightarrow VM$

$)$

$\square\; 10p \rightarrow (small \rightarrow (5pchange \rightarrow VM)$

$\square\; large \rightarrow VM$

$)\qquad)$

On its first step VM accepts either 5p or 10p. In the first case, its following step is either the acceptance of a second 5p (preparatory to withdrawal of a large packet of biscuits) or the immediate withdrawal of a small packet. The second case should be self-explanatory. In all cases, after a successful transaction, the subsequent behaviour of VM is to offer a similar service to an arbitrary long sequence of later customers. But if any customer is so unwise to put three consecutive 5p coins into the slot, the machine will break (ABORT), and never do anything else again.